

Team Notebook

JU_Jabian_6.1 | Jahangirnagar University
Jahid | Saad | Kaium

Table of Contents

1. Data Structures

2D BIT	2
BIT (Fenwick Tree)	2
Sparse Table	2
Segment Tree	2
Persistent Segment Tree	3
Merge Sort Tree	4
Trie	4

2. Graph Algorithms

DSU (Disjoint Set Union)	5
Dijkstra	5
Bellman Ford	5
Floyd Warshall	6
Articulation Point	6
Bridge	6
SCC (Kosaraju)	6
BCC (Biconnected)	7
Topological Sort	7
Cycle Detection	8
LCA (Binary Lifting)	8
Bipartite Matching	8
Max Flow	9

3. Mathematics

Sieve (Prime + SPF)	9
Euler Totient	10
Extended Euclid	10
Mobius Function	10
Inclusion-Exclusion	11
Derangements	11
Baby-step Giant-step	11
BigMod / nCr	11
Chinese Remainder Theorem	11

4. Dynamic Programming

SOS DP (Sum over Subsets)	11
Digit DP	12
Bit Mask DP	12
DP Path Print	12

5. String Algorithms

Hashing (Rolling Hash)	12
KMP	13
Z Algorithm	13
Manacher (Palindromes)	13
Suffix Array	13

6. Geometry

Basic Geometry	14
Convex Hull (Graham Scan)	14
Circle Intersection	15
Point in Polygon	15
Segment Intersection	16
Polygon Area (Shoelace)	16
Distance to Line Segment	16
Circle from 3 Points	16

7. Advanced Algorithms

Heavy-Light Decomposition	17
Mo's Algorithm	17
Square Root Decomposition	17
Euler Tour	18
Tree Diameter	18

8. 2D Algorithms

2D Prefix Sum + mat rot	18
Spiral Matrix	18
Matrix Exponentiation	18

9. Utilities

Path Compression	18
Histogram (Largest Rectangle)	18
Is Intersect All Points	19
Unique Line Equations	19

10. Special Problems

Kth Number (L to R)	19
Subarrays XOR < k	20
Min Length with Max XOR	20
Inversions $i < j < k$	20
Tetrahedron Formulas	21

11. Templates

Template with PBDS	21
GP Hash Table	21
Saad Segment Tree	21
Kaium Segment Tree	22
Kaium I/O	23
Kaium Hashing	24
Min Cut	24
Convex Hull (kaium)	25

Data Structures

2D BIT

```

1 template <typename T> class BIT2D {
2     private:
3         const int n, m;
4         vector<vector<T>> bit;
5     public:
6         BIT2D(int n, int m) : n(n), m(m), bit(n + 1, vector<T>(m + 1)) {}
7         void add(int r, int c, T val) {
8             r++, c++;
9             for (; r <= n; r += r & -r) {
10                 for (int i = c; i <= m; i += i & -i) { bit[r][i] += val; }
11             }
12         }
13         T rect_sum(int r, int c) {
14             r++, c++;
15             T sum = 0;
16             for (; r > 0; r -= r & -r) {
17                 for (int i = c; i > 0; i -= i & -i) { sum += bit[r][i]; }
18             }
19             return sum;
20         }
21         T rect_sum(int r1, int c1, int r2, int c2) {
22             return rect_sum(r2,c2)-rect_sum(r2,c1-1)-rect_sum(r1-1,c2)+
23                 rect_sum(r1-1,c1-1);
24         }
25         vector<vector<int>> ar(n,vector<int>(m)); BIT2D<int> bit(n, m); bit
            .add,bit.rect_sum

```

BIT

```

1 ll tree[N+5];
2 void update(int pos, ll val){
3     while (pos <= N) // must N = 4 * n
4     {
5         tree[pos] += val;
6         pos += (pos & -pos);
7     }
8 }
9 ll query(int pos){
10     ll sum = 0;
11     while (pos > 0)
12     {
13         sum += tree[pos];
14         pos -= (pos & -pos);
15     }
16     return sum;
17 }
18 ll sum(int l,int r){
19     return query(r) - query(l-1);

```

```

20 }

```

Sparse Table

```

1 int spt[N][22];
2 int n,ar[N];
3 int combine(int a,int b){
4     return min(a , b);
5 }
6 void buildST(){
7     for (int i = 0; i < n; i++) spt[i][0] = ar[i];
8
9     for (int j = 1; (1 << j) <= n; j++) {
10         for (int i = 0; (i + (1 << j) - 1) < n; i++) {
11             spt[i][j] = combine(spt[i + (1 << (j - 1))][j - 1] ,
12                                 spt[i][j - 1]);
13         }
14     }
15 int query(int l, int r){
16     if(l>r) return INT_MAX;
17     int j = (int)log2(r - l + 1);
18     return combine(spt[l][j], spt[r - (1 << j) + 1][j]);
19 }
20 /// 0 based index -> query 0(1),pre compute nlog(n) and support
    idempotent function f(a,a) = a
21 /// minimum, maximum, GCD, and XOR buildST();

```

Segment Tree

```

1 int ar[N],tree[4*N],lazy[4*N]; // lazy seg start
2 int combine(int x,int y){return min(x , y);}
3 void build(int node,int l,int r){
4     lazy[node] = -1; /// change
5     if(l == r){tree[node] = ar[l];return;}
6     int mid = (l + r)/2;
7     build(node*2,l,mid);build(node*2+1,mid+1,r);
8     tree[node] = combine(tree[node*2],tree[node*2 + 1]);
9 }
10 void propagate(int node,int l,int r){
11     if(lazy[node] == -1)return; /// change
12     if(l != r){lazy[2*node] = lazy[node]; /// change
13                 lazy[2*node+1] = lazy[node];}
14     tree[node] = lazy[node]; /// change
15     lazy[node] = -1; /// change
16 }
17 int query(int node,int l,int r,int a,int b){
18     propagate(node,l,r);
19     if(l > b || r < a){ return oo;}
20     if(a <= l && r <= b){return tree[node];}
21     int mid = (l + r)/2;
22     int x = query(node * 2,l,mid,a,b);
23     int y = query(node * 2 + 1,mid + 1,r,a,b);

```

```

24     return combine(x,y);
25 }
26 void update(int node,int l,int r,int a,int b,int val){
27     propagate(node,l,r); if(a > r || b < l)return;
28     if(a <= l && r <= b){
29         lazy[node] = val;/// change
30         propagate(node,l,r); return;
31     }
32     int mid = (l + r)/2; update(node*2,l,mid,a,b,val);
33     update(node*2+1,mid+1,r,a,b,val);
34     tree[node] = combine(tree[node*2],tree[node*2+1]);
35 }
36 ll tree[M],ar[M];// just seg Tree start
37 ll combine(int a,int b){return max(a,b);}
38 void init(int node,int b,int e){
39     if(b == e){tree[node] = ar[b];return;}
40     int Left = node * 2;int Right = node * 2 + 1;
41     int mid = (b+e)/2; init(Left,b,mid); init(Right,mid+1,e);
42     tree[node] = combine(tree[Left] , tree[Right]);
43 }
44 ll query(int node,int b,int e,int i,int j){
45     if(i>e || j<b)return 0;if(i<=b && j>=e){
46         return tree[node];
47     }
48     int Left = node * 2;int Right = node * 2 + 1;
49     int mid = (b+e)/2;ll p1 = query(Left,b,mid,i,j);
50     ll p2 = query(Right,mid+1,e,i,j);return combine(p1,p2);
51 }
52 void upgrade(int node,int b,int e,int i,int newValue){
53     if(b == e){tree[node] = newValue;ar[b] = newValue;
54     return;
55     }
56     int Left = node * 2;int Right = node * 2 + 1;int mid = (b+e)/2;
57     if(i <= mid)upgrade(Left,b,mid,i,newValue);
58     else upgrade(Right,mid+1,e,i,newValue);
59     tree[node] = combine(tree[Left] , tree[Right]);
60 }
61 // Iterative Segment Tree ->
62 int combine(int a,int b){return max(a,b);}
63 void build(int n){
64     for(int i=1;i<=n;i++) tree[n+i-1]=ar[i];
65     for(int i=n-1;i;i--) tree[i]=combine(tree[i<<1],tree[i<<1|1]);
66 }
67 void update(int p,int v,int n){
68     for(tree[p+=n-1]=v;p>>=1;p>>=1){
69         tree[p]=combine(tree[p<<1],tree[p<<1|1]);
70     }
71 }
72 int query(int l,int r,int n){
73     int ans=NEUTRAL;
74     for(l+=n-1,r+=n-1;l<=r;l>>=1,r>>=1){
75         if(l&1) ans=combine(ans,tree[l++]);
76         if(!(r&1)) ans=combine(ans,tree[r--]);

```

```

77     }return ans;
78 }

```

Persistent Segment Tree

```

1 struct node{
2     int val;node* lc;node* rc;
3 };
4 int ar[N];
5 node* versions[N+5];
6 vector<node*> lagbe;
7 node* build(int l,int r){
8     if(l == r){
9         node* me = new node();lagbe.pb(me);
10        me->val = ar[l]; me->lc = NULL;
11        me->rc = NULL;return me;
12    }
13    node* my = new node();lagbe.pb(my);
14    int mid = (l + r)/2;
15    my->lc = build(l,mid);
16    my->rc = build(mid+1,r);
17    my->val = (my->lc->val + my->rc->val); return my;}
18 int query(node* my,int l,int r,int L,int R){
19     if(R < l || r < L)return 0;
20     if(L <= l && r <= R){return my->val;}
21     int mid = (l + r)/2;
22     int p = query(my->lc,l,mid,L,R);
23     int q = query(my->rc,mid+1,r,L,R);
24     return p + q;}
25 node* update(node* my,int l,int r,int ind,int val){
26     if(l == r){
27         node* new_my = new node(); lagbe.pb(new_my);
28         new_my->lc = NULL;new_my->rc = NULL;new_my->val = val;
29         return new_my; }
30     node* new_me = new node();
31     lagbe.pb(new_me);
32     int mid = (l + r)/2;
33     if(ind <= mid){
34         node* x = update(my->lc,l,mid,ind,val);
35         new_me->lc = x;
36         new_me->rc = my->rc;}
37     else{
38         node* y = update(my->rc,mid+1,r,ind,val);
39         new_me->lc = my->lc;
40         new_me->rc = y;
41     }
42     new_me->val = new_me->lc->val + new_me->rc->val;
43     return new_me;
44 }
45 // after test case delete all node* ->for(auto u : lagbe){delete u;
46     }lagbe.clear();
47 int now = 1;versions[now] = build(1,n);versions[k] = update(
48     versions[k],1,n,a,x);

```

Merge Sort Tree

```

1 struct Marge_Sort_Tree {
2     int n;
3     vector<vector<int>> tree;
4     void build(vector<int> &a, int x, int l, int r) {
5         if (l + 1 == r) {
6             tree[x] = {a[l]};
7             return;
8         }
9
10        int m = (l + r) / 2;
11        build(a, 2 * x + 1, l, m);
12        build(a, 2 * x + 2, m, r);
13        merge(all(tree[2 * x + 1]), all(tree[2 * x + 2]),
14              back_inserter(tree[x]));
15    }
16    Marge_Sort_Tree(vector<int>& a) : n(a.size()) {
17        int SIZE = 1 << ( __lg(n) + bool(__builtin_popcount(n) - 1));
18        ;
19        tree.resize(2 * SIZE - 1);
20        build(a, 0, 0, n);
21    }
22    int count(int lq, int rq, int mn, int mx, int x, int l, int r)
23    {
24        if (rq <= l || r <= lq) return 0;
25        if (lq <= l && r <= rq) return lower_bound(all(tree[x]), mx
26            ) - lower_bound(all(tree[x]), mn);
27
28        int m = (l + r) / 2;
29        int a = count(lq, rq, mn, mx, 2 * x + 1, l, m);
30        int b = count(lq, rq, mn, mx, 2 * x + 2, m, r);
31        return a + b;
32    }
33    int count(int lq, int rq, int mn, int mx) {
34        return count(lq - 1, rq, mn, mx + 1, 0, 0, n); // query 1
35        base call l to r how many mn to mx number
36    }
37 };

```

Trie

```

1 struct Node {
2     Node* links[26]; int cnt_pre = 0; int cnt_word = 0;
3     bool contains(char c) { return (links[c - 'a'] != NULL); }
4     void put(char c, Node* node) { links[c - 'a'] = node; }
5     Node* get(char c) { return links[c - 'a']; }
6     void increasePre() { cnt_pre++; }
7     void increaseWord() { cnt_word++; }
8     void decreasePre() { cnt_pre--; }
9     void decreaseWord() { cnt_word--; }
10    int getPre() { return cnt_pre; }
11    int getWord() { return cnt_word; }

```

```

12    void deleteNode(){
13        for(int i = 0; i < 26; ++i) {
14            if(links[i] != nullptr) { links[i]->deleteNode();
15                delete links[i]; }
16        }
17    };
18    class Trie {
19    private:
20        Node *root;
21    public:
22        Trie() {
23            root = new Node();
24        }
25        void deleteNode(){ root->deleteNode(); delete root; }
26        void insert(string s) {
27            Node* cur_node = root;
28            for(char c : s) {
29                if(!cur_node -> contains(c)) {
30                    cur_node -> put(c, new Node());
31                } //! move reference node
32                cur_node = cur_node -> get(c);
33                cur_node -> increasePre();
34            }
35            cur_node -> increaseWord();
36        }
37        int countWords(string s) {
38            Node* cur_node = root;
39            for(char c : s) {
40                if(!cur_node -> contains(c)) {
41                    return 0;
42                }
43                cur_node = cur_node -> get(c);
44            }
45            return cur_node -> getWord();
46        }
47        int countPrefixes(string s) {
48            Node* cur_node = root;
49            for(char c : s) {
50                if(!cur_node -> contains(c)) {
51                    return 0;
52                }
53                cur_node = cur_node -> get(c);
54            }
55            return cur_node -> getPre();
56        }
57        void erase(string s) {
58            Node* cur_node = root;
59            for(char c : s) {
60                if(!cur_node -> contains(c)) {
61                    return;
62                }
63                cur_node = cur_node -> get(c);

```

```

64     cur_node -> decreasePre();
65 }
66     cur_node -> decreaseWord();
67 }
68 int getMax(int num) {
69     Node* cur_node = root;
70     int mx = 0;
71     for(int i = 31; i >= 0; i--) {
72         int bit = (num >> i) & 1;
73         if(cur_node -> contains(bit ^ 1) && cur_node -> getCnt(bit ^
74             1) > 0) {
75             mx = ((1LL << i) | mx);
76             cur_node = cur_node -> get(bit ^ 1);
77         } else {
78             cur_node = cur_node -> get(bit);
79         }
80     }
81     return mx;
82 };

```

Graph Algorithms

DSU

```

1 class DSU{
2     vector<int> parent,Size;
3 public:
4     DSU(int n){
5         for(int i = 0; i <= n; i++){
6             parent.push_back(i);
7             Size.push_back(1);
8         }
9     }
10    int find_par(int node){
11        if(parent[node] == node){
12            return node;
13        }
14        return parent[node] = find_par(parent[node]);
15    }
16    void union_size(int u, int v) {
17        int pu = find_par(u);
18        int pv = find_par(v);
19        if(pu == pv) return;
20
21        if(Size[pu] < Size[pv]){
22            parent[pu] = pv;
23            Size[pv] += Size[pu];
24        }
25        else{
26            parent[pv] = pu;
27            Size[pu] += Size[pv];

```

```

28     }
29 }
30 int size(int node){
31     return Size[find_par(node)];}
32 };

```

Dijkstra

```

1 void dijkstra(int s,int n,vi &dis,vector<vector<pii>> &graph){
2     priority_queue<pii,vector<pii>,greater<pii>> pq;
3     for(int i = 1; i <= n; i++){
4         dis[i]=oo;
5     }
6     dis[s]=0;
7     pq.push({0,s});
8     while(!pq.empty()){
9         int a = pq.top().S,co = pq.top().F;
10        pq.pop();
11        if(co > dis[a]) continue;
12        for (auto u: graph[a]){
13            int b = u.F,w = u.S;
14            if (dis[a]+w < dis[b]){
15                dis[b] = dis[a]+w;
16                pq.push({dis[b],b});
17            }
18        }
19    }
20 }

```

Bellman Ford

```

1 struct edge {
2     int u,v,w,x;
3 };
4 vector<edge> e;
5 int dist[N],par[N],neg[N];
6 void BellmenFord(int s,int n,int m) {
7     for(int i = 0; i <= n; i++) {
8         dist[i] = oo,neg[i] = 0;
9     }
10    dist[s] = 0;
11    for(int i = 1; i <= n; i++) {
12        for(int j = 0; j < m; j++) {
13            int u = e[j].u,v = e[j].v,w = e[j].w;
14            if(dist[u] + w < dist[v]) {
15                dist[v] = dist[u] + w;
16                par[v] = u;
17                if(i == n) {
18                    neg[v] = 1;
19                }
20            }
21        }
22    }

```

```
23 }// neg[i] tells it is a part of negative cycle or not
```

Floyd Warshall

```
1 for(int i = 1; i <= n; i++) {
2     for(int j = 1; j <= n; j++) {
3         nxt[i][j] = j;
4     }
5 }
6 for(int k = 1; k <= n; k++) {
7     for(int i = 1; i <= n; i++) {
8         for(int j = 1; j <= n; j++) {
9             if(dist[i][k] + dist[k][j] < dist[i][j]) {
10                 dist[i][j] = dist[i][k] + dist[k][j];
11                 nxt[i][j] = nxt[i][k];
12             }
13         }
14     }
15 }
16 int s = 1, t = 4;
17 cout << s << " ";
18 int cur = s;
19 while(cur != t) {
20     cur = nxt[cur][t];
21     cout << cur << " ";
22 }
```

Articulation Point

```
1 vi graph[N];set<int> point;
2 int low[N],tin[N],vis[N],timer;
3 void dfs(int node,int par){
4     vis[node] = 1;
5     low[node] = tin[node] = ++timer;
6     int child = 0;
7     for(auto u : graph[node]) {
8         if(u == par)continue;
9         if(vis[u])low[node] = min(low[node],tin[u]);
10        else {
11            child++;
12            dfs(u,node);
13            low[node] = min(low[node],low[u]);
14            if(low[u] >= tin[node] && par != -1)point.insert(node);
15        }
16    }
17    if(child > 1 && par == -1)point.insert(node);
18 }
```

Bridge

```
1 set<pii> bridge;
2 int low[N],tin[N],vis[N],timer = 0;
3 void dfs(int node,int par){
```

```
4     vis[node] = 1;
5     low[node] = tin[node] = ++timer;
6     for(auto child : graph[node]) {
7         if(child == par)continue;
8         if(vis[child]) {
9             low[node] = min(low[node], tin[child]);
10        }
11        else {
12            dfs(child,node);
13            low[node] = min(low[node], low[child]);
14            if(low[child] > tin[node]) {
15                int x = min(node,child),y = max(node,child);
16                bridge.insert({x,y});
17            }
18        }
19    }
20 }
```

SCC

```
1 // given a directed graph return the minimum number of edges to be
   added so that the whole graph become an SCC
2 bool vis[N];
3 vector<int> g[N], r[N], G[N], vec; //g is the condensed graph
4 void dfs1(int u) {
5     vis[u] = 1;
6     for(auto v: g[u]) if(!vis[v]) dfs1(v);
7     vec.push_back(u);
8 }
9 vector<int> comp;
10 void dfs2(int u) {
11     comp.push_back(u);
12     vis[u] = 1;
13     for(auto v: r[u]) if(!vis[v]) dfs2(v);
14 }
15 int idx[N], in[N], out[N];
16 int main() {
17     int n, m;
18     cin >> n >> m;
19     for(int i = 1; i <= m; i++) {
20         int u, v;
21         cin >> u >> v;
22         g[u].push_back(v);
23         r[v].push_back(u);
24     }
25     for(int i = 1; i <= n; i++) if(!vis[i]) dfs1(i);
26     reverse(vec.begin(), vec.end());
27     memset(vis, 0, sizeof vis);
28     int scc = 0;
29     for(auto u: vec) {
30         if(!vis[u]) {
31             comp.clear();
32             dfs2(u);
```

```

33     scc++;
34     for(auto x: comp) idx[x]=scc;
35 }
36 }
37 for(int u = 1; u <= n; u++) {
38     for(auto v: g[u]) {
39         if(idx[u] != idx[v]) {
40             in[idx[v]]++, out[idx[u]]++;
41             G[idx[u]].push_back(idx[v]);
42         }
43     }
44 }
45 int needed_in=0, needed_out=0;
46 for(int i = 1; i <= scc; i++) {
47     if(!in[i]) needed_in++;
48     if(!out[i]) needed_out++;
49 }
50 int ans = max(needed_in, needed_out);
51 if(scc == 1) ans = 0;
52 cout << ans << '\n';
53 }

```

BCC

```

1 /bcc[i] is which num Biconnected element it
2 set<int> g[N]; //use set avoid multiple same edges
3 vector<int> bcc[N], st;
4 int low[N], dis[N], T = 0, sz = 0;
5 vector<vector<int>> allBCC;
6 void dfs(int u, int pre = -1) {
7     low[u] = dis[u] = ++T;
8     st.push_back(u);
9     for(auto v : g[u]) {
10         if(!dis[v]) {
11             dfs(v, u);
12             low[u] = min(low[u], low[v]);
13             if(low[v] >= dis[u]) {
14                 sz++; vector<int> comp;
15                 int x;
16                 do{
17                     x = st.back();
18                     st.pop_back();
19                     bcc[x].push_back(sz);
20                     comp.push_back(x);
21                 } while (x != v);
22                 bcc[u].push_back(sz);
23                 comp.push_back(u);
24                 sort(comp.begin(), comp.end());
25                 comp.erase(unique(comp.begin(), comp.end()), comp.
26                     end());
27                 allBCC.push_back(comp);
28             }
29         }
30     }
31 }

```

```

29         else if(v != pre)
30             low[u] = min(low[u], dis[v]);
31     }
32 }
33 void Goriber_solve(){
34     int n,m; cin >> n >> m;
35     for (int i = 0; i < m; i++) {
36         int u, v; cin >> u >> v;
37         if (u == v) continue;
38         g[u].insert(v); g[v].insert(u);
39     }
40     for(int i = 0; i < n; i++) if(!dis[i]) dfs(i);
41     for (int i = 0; i < n; i++) {
42         if(g[i].empty()) {sz++;
43             allBCC.push_back({i});
44         }
45     }
46     cout << allBCC.size() << "\n";
47     for(auto &comp : allBCC) {
48         cout << comp.size();
49         for(int x : comp) cout << " " << x;
50         cout << "\n";
51     }
52 } //if bcc[u].size() > 1 it is an AP

```

Topological Sort

```

1 void dfs(int node) {
2     vis[node] = 1;
3     for(auto u : graph[node]) {
4         if(!vis[u]) { dfs(u); }
5     } ans.pb(node);
6 }
7 void Goriber_solve() {
8     int n,m; cin >> n >> m; bool ok = 0;
9     for(int i = 0; i < m; i++) {
10         int u,v; cin >> u >> v;
11         if(u == v) ok = 1; graph[u].pb(v);
12     }
13     if(ok) {cout << "IMPOSSIBLE\n"; return;}
14     for(int i = 1; i <= n; i++) {
15         if(!vis[i]) {dfs(i);}
16     }
17     reverse(all(ans)); vi pos(n+1);
18     for(int i = 0; i < sz(ans); i++) {
19         pos[ans[i]] = i+1;
20     }
21     for(int i = 1; i <= n; i++) {
22         for(auto u : graph[i]) {
23             if(pos[i] > pos[u]) {cout << "IMPOSSIBLE\n"; return;}
24         }
25     }
26 }

```

```

27     for(auto u : ans) {cout << u << " ";}cout << endl;
28 }

```

Cycle Detection

```

1 vi graph[N]; // for directed graph
2 int vis[N], start = -1, finish, par[N];
3 void dfs(int node){
4     vis[node] = 1;
5     for(auto child : graph[node]){
6         if(vis[child] == 0){
7             par[child] = node;
8             dfs(child);
9             if(start != -1) return;
10        }
11        else if(vis[child] == 1){
12            start = child;
13            finish = node;
14            return;
15        }
16    }
17    vis[node] = 2;
18 }
19 bool is_cycle(int node, int par) { // for undirected graph
20     vis[node] = true;
21     ans.pb(node);
22     for(int child : graph[node]) {
23         if(child == par) continue;
24         if(vis[child] == 0) {
25             if(is_cycle(child, node) == 1) return 1;
26         }
27         else {
28             ans.pb(child);
29             return 1;
30         }
31     }
32     ans.pop_back();
33     return 0;
34 }
35 //main function->
36 for(int i = 1; i <= n; i++){
37     if(vis[i] == 0){dfs(i); if(start != -1) break;}
38     int x = ans.back(); vi path; path.pb(x); ans.pop_back(); int node = ans
39         .back();
40     while(node != x) {
41         path.pb(node); ans.pop_back(); node = ans.back();
42     } path.pb(x);

```

LCA

```

1 vector<int> graph[N];
2 int Level[N]; int dp[N][22];
3 void dfs(int node, int par, int lev){

```

```

4     dp[node][0] = par;
5     Level[node] = lev;
6     for(int i = 1; i <= 20; i++){
7         dp[node][i] = dp[dp[node][i-1]][i-1];
8     }
9     for(auto child : graph[node]){
10        if(child != par){
11            dfs(child, node, lev+1);
12        }
13    }
14 }
15 int get_kth_par(int node, int k){
16     for(int i = 20; i >= 0; i--){
17         if((k >> i) & 1){
18             node = dp[node][i];
19         }
20     } return node;
21 }
22 int get_lca(int a, int b){
23     if(Level[a] > Level[b]){
24         swap(a, b);
25     }
26     int k = Level[b] - Level[a];
27     b = get_kth_par(b, k);
28     if(a == b) return a;
29     for(int i = 20; i >= 0; i--){
30         if(dp[a][i] != dp[b][i]){
31             a = dp[a][i];
32             b = dp[b][i];
33         }
34     } return dp[a][0];
35 } //dfs(1,0,0);

```

Bipartite Matching

```

1 const int N = 3e5 + 9; // Time complexity O(E*sqrt(V))
2 struct HopcroftKarp {
3     static const int inf = 1e9; int n;
4     vector<int> l, r, d; vector<vector<int>>> g;
5     HopcroftKarp(int _n, int _m) {
6         n = _n; int p = _n + _m + 1; g.resize(p);
7         l.resize(p, 0); r.resize(p, 0); d.resize(p, 0);
8     }
9     void add_edge(int u, int v) {
10        g[u].push_back(v + n); //right id is increased by n, so is l[u]
11    }
12    bool bfs() {
13        queue<int> q;
14        for (int u = 1; u <= n; u++) {
15            if (!l[u]) {d[u] = 0; q.push(u);}
16            else d[u] = inf;
17        } d[0] = inf;
18        while (!q.empty()) {

```



```

19     int u = q.front();q.pop();
20     for (auto v : g[u]) {
21         if (d[r[v]] == inf) {
22             d[r[v]] = d[u] + 1;q.push(r[v]);
23         }
24     }
25 }
26 return d[0] != inf;
27 }
28 bool dfs(int u) {
29     if (!u) return true;
30     for (auto v : g[u]) {
31         if(d[r[v]] == d[u] + 1 && dfs(r[v])) {
32             l[u] = v;r[v] = u;return true;
33         }
34     } d[u] = inf;
35     return false;
36 }
37 int maximum_matching() {
38     int ans = 0;
39     while (bfs()) {
40         for(int u = 1; u <= n; u++)if(!l[u] && dfs(u)) { ans++; }
41     } return ans;
42 }
43 vector<pii> get_pairs() {
44     vector<pii> pairs;
45     for(int u = 1; u <= n; u++) {
46         if(l[u]) {
47             pairs.emplace_back(u,l[u] - n);
48         }
49     } return pairs;
50 }
51 };

```

Max Flow (Dinic)

```

1 const int N = 5010; const long long inf = 1LL << 61;
2 struct Dinic {
3     struct edge {
4         int to, rev;long long flow, w;int id;
5     };
6     int n, s, t, mxid;vector<int> d, flow_through;
7     vector<int> done; vector<vector<edge>> g;
8     Dinic() {}
9     Dinic(int _n) {
10         n = _n + 10;mxid = 0;g.resize(n);
11     }
12     void add_edge(int u, int v, long long w, int id = -1) {
13         edge a = {v, (int)g[v].size(), 0, w, id};
14         edge b = {u, (int)g[u].size(), 0, 0, -2};//for bidirectional
15         edges cap(b) = w
16         g[u].emplace_back(a);g[v].emplace_back(b);mxid = max(mxid, id);

```

```

17     bool bfs() {
18         d.assign(n, -1);d[s] = 0;queue<int> q;q.push(s);
19         while (!q.empty()) {
20             int u = q.front();q.pop();
21             for (auto &e : g[u]) {
22                 int v = e.to;
23                 if (d[v] == -1 && e.flow < e.w) d[v] = d[u] + 1, q.push(v);
24             }
25         } return d[t] != -1;
26     }
27     long long dfs(int u, long long flow) {
28         if (u == t) return flow;
29         for (int &i = done[u]; i < (int)g[u].size(); i++) {
30             edge &e = g[u][i]; if (e.w <= e.flow) continue;
31             int v = e.to;
32             if (d[v] == d[u] + 1) {
33                 long long nw = dfs(v, min(flow, e.w - e.flow));
34                 if (nw > 0) {
35                     e.flow += nw;g[v][e.rev].flow -= nw;
36                     return nw;
37                 }
38             }
39         }return 0;
40     }
41     long long max_flow(int _s, int _t) {
42         s = _s;t = _t;long long flow = 0;
43         while (bfs()) {
44             done.assign(n, 0);
45             while (long long nw = dfs(s, inf)) flow += nw;
46         }
47         flow_through.assign(mxid + 10, 0);
48         for(int i = 0; i < n; i++) for(auto e : g[i]) {
49             if(e.id >= 0) flow_through[e.id] = e.flow;
50         } return flow;
51     }
52     void reset_flows() {
53         for (int i = 0; i < n; ++i) {
54             for (auto& e : g[i]) {
55                 e.flow = 0;
56             }
57         }
58     }
59 };

```

Mathematics

Sieve

```

1 // use bitset<N> is_prime; to have O(N/64) memory complexity
2 // using bitset you can solve upto around N = 10^8 in 1s
3 const int N = 1e8;vector<int> primes;bitset<N>is_prime;
4 void sieve_v0() {

```

```

5  for (int i = 3; i < N; i += 2) {
6      is_prime[i] = 1;
7  }
8  for (int i = 3; i * i < N; i += 2) {
9      if (is_prime[i]) {
10         for (int j = i * i; j < N; j += i) {
11             is_prime[j] = 0;
12         }
13     }
14 }
15 is_prime[2] = 1;
16 for (int i = 2; i < N; i++) {
17     if(is_prime[i]){ primes.push_back(i); }
18 }
19 } // sieve with smallest prime factors (spf)
20 int spf[N];
21 void sieve() {
22     for (int i = 2; i < N; i++) { spf[i] = i; }
23     for (int i = 2; i * i < N; i++) {
24         if (spf[i] == i) {
25             for (int j = i * i; j < N; j += i) {
26                 spf[j] = min(spf[j], i);
27             }
28         }
29     }
30     for (int i = 2; i < N; i++) {
31         if (spf[i] == i) {
32             primes.push_back(i);
33         }
34     }
35 }
36 sum of divisor = ((x^(a-1)/(x - 1)) * (y ^ (b-1)/ (y-1)));

```

Euler Totient

```

1  int phi[N];bool mark[N];
2  void sieve(){
3      for(int i = 1; i < N; i++){ phi[i] = i; }
4      for(int i = 2; i < N; i++){
5          if(mark[i])continue; phi[i] = i - 1;
6          for(int j = i + i; j < N; j += i){
7              mark[j] = true; phi[j] -= phi[j] / i;
8          }
9      }
10 }
11 long long euler_phi(long long x){
12     long long ans = x;
13     for (long long i = 2; i * i <= x; i++){
14         if (x % i == 0){
15             ans /= i; ans *= i - 1;
16             while (x % i == 0){ x /= i; }
17         }
18     }

```

```

19     if (x > 1){
20         ans /= x;ans *= x - 1;
21     }
22     return ans;
23 }
24 1-> f(n) = gcd(1,n)+gcd(2,n)+...+gcd(n,n)
25 ans,f(n) = sum{divisor of n as d} d * phi(n/d)

```

Extended Euclid

```

1  struct triplate { int x,y,gcd;};
2  /// ax + by = gcd(a,b)
3  triplate extendedEuclid(int a, int b) {
4      if(b == 0)return {1, 0, a};
5      triplate smallans = extendedEuclid(b, a % b);
6      triplate ans; ans.x = smallans.y; ans.gcd = smallans.gcd;
7      ans.y = (smallans.x - (a / b) * smallans.y);
8      return ans;
9  }// x^-1 = power(x,phi(mod)-1,mod) power(a,b,mod) = (a ^ (b%phi(mod)
   )))%mod;

```

Mobius

```

1  void mobius() { mu[1] = 1;
2      for(int i = 2; i <= N; i++) {
3          if(!isComposite[i]) {
4              primes.push_back(i);
5              mu[i] = -1;
6          }
7          for(int p : primes) {
8              if(1LL * i * p > N)break;
9              isComposite[i * p] = 1;
10             if(i % p == 0) {mu[i * p] = 0;break;}
11             else {mu[i * p] = -mu[i];}
12         }
13     }
14 }
15 }
16 1-> Number of unordered pairs(i, j) with gcd(a_i, a_j) = 1 freq[x]
   = how many array elements equal x
17 cnt[d] = number of array elements divisible by d,pair count = sum{1
   to (mx of ar) as d} mu[d] * nCr(cnt[d],2)
18 subsequence count = sum{1 to mx} mu[d] * (pow(2,cnt[d])-1)
19 2->How many pair range[1..N] gcd(x,y) = 1 ,ans = sum{1 to N as d}
   mu[d]*(N/d)^2
20 3-> Count Numbers <= N that are divisable by none of primes
21 L = lcm of all prime.. divs = all divisor of L,ans = sum{divs as d}
   mu[d]*(N/d)
22 4-> How many squire free num in [1..N] count = sum{1 to sqrt(N) as d
   } mu[d]*(N/(d*d))
23 5-> how many Lattice point with gcd = 1 or 1 <= x <= n and 1 <= y
   <= m and gcd(x,y) = 1
24 ,ans = sum{1 to min(x,y) as d} mu[d]*(n/d)*(m/d)

```

Inclusion-Exclusion

```

1 int ans = n, la = (1 << k);
2 for(int i = 1; i < la; i++) { int lc = 1;
3     for(int j = 0; j < k; j++) {
4         if(getbit(i,j)) { lc = lcm(lc,ar[j]);if(lc > n)break;
5     }
6 } int cnt = __builtin_popcountll(i);
7 if(cnt & 1)ans -= (n / lc);
8 else ans += (n / lc);
9 }

```

Derangements find n

```

1 int Derangements(int n) { // 0(n)
2     vector<int> dp(n + 1); dp[1] = 0; dp[2] = 1;
3     for (int i = 3; i <= n; i++) {
4         dp[i] = (i - 1) * (dp[i - 1] + dp[i - 2]);
5     }return dp[n];
6 }
7 int find_n(int x) { // n*(n+1)/2 = x
8     int n = (-1 + sqrt(1 + 8 * x)) / 2;
9     if((n * (n + 1)) / 2 > x) { n--; }
10    return n;
11 }

```

Baby-step Giant-step

```

1 //Returns minimum x for which a ^ x % m = b % m
2 int baby_step(int a,int b,int m) {
3     a%=m,b%=m;int k=1,add=0,g;
4     while((g=__gcd(a,m))>1) {
5         if(b==k)return add;if(b%g)return -1;
6         b/=g,m/=g,add++;k=(k*a/g)%m;
7     }
8     int n=sqrt(m)+1;int mul=1;
9     for(int i=0; i<n; i++)mul=(mul*a)%m;
10    unordered_map<int,int>val;
11    for(int q=0,cur=b; q<=n; q++) {
12        val[cur]=q;cur=(cur*a)%m;
13    }
14    for(int p=1,cur=k; p<=n; p++) {
15        cur=(cur*mul)%m;
16        if(val.count(cur)) {return p*n-val[cur]+add;}
17    } return -1;
18 }

```

BigMod/nCr

```

1 int power(int a,int b,int mod) {
2     ll ans = 1;
3     while(b>0) {
4         if(b&1) { ans = (ans*a)%mod;

```

```

5         a = (a*a)%mod;b >>= 1;
6     } return ans;
7 }
8 int fact[N];
9 void Pre() {
10    fact[0] = 1;
11    for(int i = 1; i < N; i++) { fact[i] = (fact[i-1] * i) % mod; }
12 }
13 int nCr(int n,int r) {
14    if(n < r)return 0ll; ll up = fact[n];
15    int down = (fact[r]*fact[n-r]) % mod;
16    down = power(down,mod-2,mod);ll res = (up * down) % mod;
17    return res;
18 }
19 Catalan Number Cn = (1/(n+1)*nCr(2n,n); Or Cn = nCr(2*n,n) - nCr(2*
    n,n-1);

```

Chinese Remainder Theorem

```

1 int ex_gcd(int a,int b,int &x,int &y) {
2     if(b == 0) { x = 1; y = 0; return a;}
3     int x1,y1; int ans = ex_gcd(b,a % b,x1,y1);
4     x = y1; y = x1 - y1 * (a / b); return ans;
5 }
6 int chinese_rem_theorem(int a,int m1,int b,int m2) {
7     int p,q; int gc = ex_gcd(m1,m2,p,q);
8     return (a*m2*q % (m1*m2) + b*m1*p % (m1*m2)) % (m1*m2);
9 }
10 void Goriber_solve() {
11     int n; cin >> n; vii a(n);
12     for(int i = 0; i < n; i++) { cin >> a[i].F >> a[i].S;}
13     int x = chinese_rem_theorem(a[0].F,a[0].S,a[1].F,a[1].S);
14     int mm = lcm(a[0].S,a[1].S);
15     for(int i = 2; i < n; i++) {
16         x = chinese_rem_theorem(x,mm,a[i].F,a[i].S); mm = lcm(mm,a[i].S
17     );
18 } cout << x << endl;

```

Dynamic Programming

SOS DP

```

1 const ll N = 20; const ll M = (1 << N);
2 void add_super(int dp[]){
3     for(int bit = 0; bit < N; bit++){
4         for(int i = 0; i < M; i++){
5             if(i & (1 << bit)){ dp[i] += dp[i^(1<<bit)];}
6         }
7     }
8 }
9 void remove_super(int dp[]){

```

```

10     for(int bit = 0; bit < N; bit++){
11         for(int i = M-1; i >= 0; i--){
12             if(i & (1 << bit)){ dp[i] -= dp[i^(1<<bit)];}
13         }
14     }
15 }
16 void add_sub(int dp[]){
17     for(int bit = 0; bit < N; bit++){
18         for(int i = M - 1; i >= 0; i--){
19             if(i & (1 << bit)){dp[i^(1<<bit)] += dp[i];}
20         }
21     }
22 }
23 void remove_sub(int dp[]){
24     for(int bit = 0; bit < N; bit++){
25         for(int i = 0; i < M; i++){
26             if(i & (1 << bit)){dp[i^(1<<bit)] -= dp[i];}
27         }
28     }
29 }
30 1.number of element y such that x|y=x ->add_super
31 2.number of element y such that x&y=x ->add_sub
32 3.number of element y such that x&y=0 ->add_super,cnt(2^bit-1)^x
33 4.(a1&a2&a4&a5&..&ax) = k -> add_sub(cnt[]), cnt[i]=pow(2,cnt[i])
    -1, remove_sub(cnt[]);

```

Digit DP

```

1 vi digit;int dp[11][100][100][3];
2 int sol(int i,int m1,int m2,int f) {
3     if(i == n) { return m1 == 0 && m2 == 0;}
4     int &ret = dp[i][m1][m2][f];if(ret != -1)return ret;
5     int res = 0;int lim = (f == 1 ? 9 : digit[i]);
6     for(int j = 0; j <= lim; j++) {
7         res += sol(i+1,(m1*10 + j) % k, (m2 + j) % k,(f || j < digit[i]
8         ));
9     }return ret = res;
10 }
11 int cal(int val) {
12     digit.clear();
13     while(val > 0) {
14         digit.push_back(val % 10);val /= 10;
15     }
16     reverse(all(digit));n = sz(digit);mem(dp,-1);
17     return sol(0,0,0,0);
18 }
19 void solve() {
20     cin >> l >> r >> k;if(k > 90)cout << 0 << endl;
21     else cout << cal(r) - cal(l-1) << endl;
22 }

```

Bit Mask DP

```

1 int f(int mask){
2     if(mask == ((1 << n) - 1))return 0; int &ret = dp[mask];
3     if(ret != -1)return ret;int res = INF;
4     for(int i = 0; i < n; i++) {
5         if((mask & (1 << i)) == 0) { int c = cost[i][i];
6             for(int j = 0; j < n; j++) {
7                 if(mask & (1 << j)) { c += cost[i][j];}
8             }
9             res = min(res,c + f(mask | (1 << i)));
10        }
11    }ret = res;
12 }

```

DP Path Print

```

1 int f(int i,int rem) {
2     if(i == n + 1) {if(rem == 0)return 0;return -n;}
3     int &ret = dp[i][rem];
4     if(ret != -1)return ret;
5     return ret = max(1 + f(i+1,(ar[i]+rem) % k),f(i+1,rem));
6 }
7 void print(int i,int rem) {
8     if(i == n + 1 || ans == 0) { return; }
9     if(1 + f(i+1,(ar[i]+rem)%k) == ans) {
10        cout << i << " "; ans--;
11        print(i+1,(rem + ar[i])%k);
12    }
13    else print(i+1,rem);
14 }

```

String Algorithms

Hashing

```

1 struct rH{
2     const int k = 239,mod = 1000000207;string s;int n;vi pw,vl;
3     rH(string s): s(s),n(sz(s)),pw(n+1),vl(n+1){ pw[0] = 1,vl[0] =
4         0;
5         for(int i = 0; i < n; i++){
6             vl[i+1] = (1LL * vl[i] * k % mod + s[i]) % mod; pw[i+1]
7             = (1LL*pw[i] * k) % mod;
8         }
9         int h(int l){ return vl[l]; }
10        int h(int l,int r){ return (h(r+1) - (1LL*h(l) * pw[r-l+1]) %
11        mod + mod) % mod;}
12 };
13 int k = 1201 ,mod = 1000003891;k = 31,mod = 1000003247;rH hs1(s);
14 cin >> l >> r;
15 l--,r--;int val = hs.h(l,r);int x = (n - l - 1);int y = x - (r - l)
16 ;int val2 = hs2.h(y,x);

```

KMP

```

1 vi build_lps(string p) {
2     int sz = p.size();vi lps;lps.assign(sz + 1, 0);int j = 0;lps[0] =
    0;
3     for(int i = 1; i < sz; i++) {
4         while(j >= 0 && p[i] != p[j]) {
5             if(j >= 1) j = lps[j - 1]; else j = -1;
6         }j++;lps[i] = j;
7     }return lps;
8 }vi ans;
9 void kmp(vi lps, string s, string p) {
10     int psz = p.size(), sz = s.size(), j = 0;
11     for(int i = 0; i < sz; i++) {
12         while(j >= 0 && p[j] != s[i])
13             if(j >= 1) j = lps[j - 1];
14             else j = -1;
15         j++;
16         if(j == psz) {j = lps[j - 1];ans.pb(i - psz + 1); }
17     }
18 }

```

Z Algorithm

```

1 // An element Z[i] of Z array stores length of the longest
    substring
2 // starting from str[i] which is also a prefix of str[0..n-1].
3 // The first entry of Z array is meaning less as complete string is
    always prefix of itself.Here Z[0]=0.
4 vector<int> z_function(string s) {
5     int n = (int) s.length();vector<int> z(n);
6     for(int i = 1, l = 0, r = 0; i < n; ++i){
7         if(i <= r){ z[i] = min (r - i + 1, z[i - l]);}
8         while (i + z[i] < n && s[z[i]] == s[i + z[i]]){++z[i];}
9         if(i + z[i] - 1 > r){ l = i, r = i + z[i] - 1; }
10    }return z;
11 }

```

Manacher

```

1 struct manacher { vector<int> p;
2     void run_manacher(string s) {
3         int n = sz(s);p.assign(n, 1);int l = 1,r = 1;
4         for(int i = 1; i < n; i++) {
5             p[i] = max(OLL,min(r - i,p[l + r - i]));
6             while(i + p[i] < n && i - p[i] >= 0 && s[i + p[i]] == s[i - p
                [i]]) {
7                 p[i]++;
8             }
9             if(i + p[i] > r) {
10                 l = i - p[i];r = i + p[i];
11             }
12     }

```

```

13     }
14     void build(string s) {
15         string t = "#";for (char c : s) t += c, t += "#";run_manacher(t
            );
16     }
17     int get_longest(int cen,bool odd) {
18         int pos = 2 * cen + 1 + (!odd);return p[pos] - 1;
19     }
20     bool is_palindrome(int l,int r) {
21         if((r - l + 1) <= get_longest((l + r) / 2,l % 2 == r % 2))
            return 1;
22         else return 0;
23     }
24 };manacher m;m.build(s);

```

Suffix Array

```

1 #define MAX_N 500010
2 string T; int RA[MAX_N],tempRA[MAX_N],n;
3 int SA[MAX_N],tempSA[MAX_N],c[MAX_N];int Phi[MAX_N],PLCP[MAX_N],LCP
    [MAX_N];
4 void countingSort(int k){
5     int i, sum, maxi = max(300, n);memset(c, 0, sizeof c);
6     for(i=0;i<n;i++)c[i+k<n?RA[i+k]:0]++;
7     for(i = sum = 0; i < maxi; i++){
8         int t = c[i];c[i] = sum;sum += t;
9     }
10    for(i = 0; i < n; i++){tempSA[c[SA[i]+k < n ? RA[SA[i]+k] :
        0]++] = SA[i];}
11    for(i = 0; i < n; i++)SA[i]=tempSA[i];
12 }
13 void constructSA(){
14     int i, k, r;for(i = 0; i < n; i++)RA[i] = T[i];
15     for(i = 0; i < n; i++)SA[i] = i;
16     for(k = 1; k < n; k <= 1){
17         countingSort(k);countingSort(0);tempRA[SA[0]] = r = 0;
18         for(i = 1; i < n; i++){tempRA[SA[i]] =(RA[SA[i]]==RA[SA[i
                -1]] && RA[SA[i]+k]==RA[SA[i-1]+k])?r:++r;}
19         for(i = 0; i < n; i++)RA[i] = tempRA[i]; if(RA[SA[n-1]] ==
            n-1)break;
20     }
21 }
22 void computeLCP() {
23     int i, L;Phi[SA[0]] = -1;for(i = 1; i < n; i++)Phi[SA[i]] = SA[
        i-1];
24     for(i = L = 0; i < n; i++){
25         if(Phi[i] == -1) {PLCP[i] = 0;continue;}while(T[i + L] == T
            [Phi[i] + L])L++;
26         PLCP[i] = L;L = max(L-1, 0);
27     }for(i = 0; i < n; i++)LCP[i] = PLCP[SA[i]];
28 }// main function code ->
29 cin >> T;T.pb('$');n = sz(T);constructSA();computeLCP();
30 1.Substring Search->Binary Search on Suffix array

```

```

31 2.Counting Substrings->Binary serach SA Upper-Lowerbound
32 3.Distrinct Substrings->(n*(n+1))/2 - sumOf(LCP)
33 4.Longest Common Substring->T+'#'+Pat+'$',SAi < n1 && SAj > n1
    maxOf LCP
34 5.Finding Longest Substring that occur >= K time-> LCP K-1
    consicutive = k time max is length of Substring
35 6.Kth smallest substring->suflen = n-SA[i]-1,i=1 to n-1,new_substr=43
    suflen-LCP[i],if new<=k,len=LCP[i]+k,start=SA[i]

```

Geometry

Basic Geometry

```

1 struct pt {
2     dl x, y;pt() : x(0), y(0) {} pt(dl a, dl b) : x(a), y(b) {}
3 };// Function to calculate the cross product (direction) of vectors
    (p1p2) and (p1p3)
4 int direction(const pt &p1, const pt &p2, const pt &p3) {
5     return (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x -
        p1.x);
6 }// Function to check if pt p3 lies on the segment p1p2
7 bool onsegment(const pt &p1, const pt &p2, const pt &p3) {
8     return min(p1.x, p2.x) <= p3.x && p3.x <= max(p1.x, p2.x) &&
9         min(p1.y, p2.y) <= p3.y && p3.y <= max(p1.y, p2.y);
10 }
11 pt rotateCCW(pt a,dl ang){
12     pt v = {a.x,a.y};dl c = cos(ang),s = sin(ang);
13     pt res = {v.x*c-v.y*s,v.x*s+v.y*c};
14     return res;}
15 pt rotateCCW(pt a,pt b,dl ang){
16     pt v = {b.x-a.x,b.y-a.y};// vec from a to b
17     dl c = cos(ang),s = sin(ang);// ang is rediun
18     pt res = {v.x*c-v.y*s,v.x*s+v.y*c};
19     return res;//rotateCW is (-s put);}
20 pt unit(pt a){
21     double val = sqrt(a.x*a.x+a.y*a.y); return pt(a.x/val,a.y/val);
22 }
23 pt vec(pt a, pt b){
24     return {b.x - a.x, b.y - a.y};}
25 dl sqnorm(pt a){
26     return (a.x*a.x + a.y*a.y);}
27 //given point a,b and angle abc and len of bc then find point c
28 pt find_point(pt a,pt b,dl ang,dl len){
29     pt c = vec(b,a); c = unit(c);
30     c.x *= len, c.y *= len;
31     c = rotateCCW(c,ang);
32     c.x += b.x, c.y += b.y;
33     return c;}
34 pt find_point(pt b,dl ang,dl len){//same as up
35     pt c = {b.x + len*cos(ang),b.y + len*sin(ang)};
36     return c;// here ang is CCW , if CW it PI - ang}
37 dl dot(pt a, pt b){

```

```

37     return (a.x*b.x + a.y*b.y);}
38 dl angle(pt a,pt o,pt b){
39     pt oa = vec(o , a), ob = vec(o , b);
40     if(sqnorm(oa) == 0 || sqnorm(ob) == 0){
41         return 0;
42     }
43     double alpha = acos(dot(oa, ob) / sqrt(sqnorm(oa) * sqnorm(ob))
44         );
45     alpha = alpha * 180 / PI;//angle in degree
46     return alpha;// angle of AOB}
47 double abs(pt a) {return sqrt(sqnorm(a));}
48 int triArea2(const pt &a, const pt &b, const pt &c) {
49     return (a.x*(b.y-c.y) + b.x*(c.y-a.y) + c.x*(a.y-b.y));
50 }
51 dl sqDist(const pt &a, const pt &b) {
52     return ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
53 }
54 dl angle(dl a, dl b,dl c){
55     dl alpha = acos((b*b-c*c-a*a)/(-2*a*c));//alpha = alpha*180.0/
56     PI;//angle (ABC) in degree
57     return alpha;}
58 dl Slope(pt a, pt b){
59     if(a.x == b.x && a.y == b.y)return oo;
60     double dx = b.x - a.x;
61     double dy = b.y - a.y;
62     if(dx == 0)return oo;
63     return (double)dy / dx;}
64 Basic Law > circle arc area = 0.5*r*r*theta ,radius of incircle r
    = A/s
65 radius of circumcircle R=(a*b*c)/(4*A),a spherical cap with height
    H and R = H/2
66 dl volume = (PI * h * h/ 3.0) * (3.0 * R - h);Polygone n point and
    same distance s
67 dl area_polygone=(n*s*s)/(4*(tan(PI/n)));dl redius_polygone = s /
    (2*sin(PI / n));
68 glass area = ((pi*h)/3) *((r1*r1)+(r2*r2)+(r1*r2)) ;N inner circle
    in One circle R
69 dl theta = PI / n;dl r = (R * sin(theta)) / (1 + sin(theta));dl
    area = PI*r*r;
70 dl E = (R - r);dl AreaNotcoverInner = ((n * r * r) / tan(theta)) -
    (area * (n - 2)) / 2.0;
71 dl AreaNotcoverOuter = Area - n * area - AreaNotcoverInner;
72 Tseq = (2^N - 1) = Even Subsequence + OddSubsequence ;Eseq = 2^Even
    ;
73 Oseq = 2^(Odd - 1);OddSumSeq = (2^Even * (2^(Odd - 1)));EvenSumSeq
    = ((2^N) - 1 - OddSumSeq);
74 Odd= Total number of odd in the array.;Oseq=Total number of
    subsequences with only odd;
75 Num ;OddSumSeq = Total number of subsequences with odd sum

```

Convex Hull

```

1 ConvexHull : Grahams Scan O(n lg n), integer implementation ;P[]:

```

```

    holds all the points, C[]: holds points on the hull
2 np: number of points in P[], nc: number of points in C[]
3 to handle duplicate, call makeUnique() before calling convexHull()
    call convexHull() if you have np >= 3
4 to remove co-linear points on hull, call compress() after
    convexHull()
5 struct point{ int x, y; };
6 point P[N], C[N], P0;
7 int triArea2(const point &a, const point &b, const point &c) {
8     return (a.x*(b.y-c.y) + b.x*(c.y-a.y) + c.x*(a.y-b.y));
9 }
10 int sqDist(const point &a, const point &b) {
11     return ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
12 }
13 bool comp(const point &a, const point &b) {
14     int d = triArea2(P0, a, b); if(d < 0) return false;
15     if(!d && sqDist(P0, a) > sqDist(P0, b)) return false;
16     return true;
17 }
18 bool normal(const point &a, const point &b) {
19     return ((a.x==b.x) ? a.y < b.y : a.x < b.x);
20 }
21 bool issame(const point &a, const point &b) {
22     return (a.x == b.x && a.y == b.y);
23 }
24 void makeUnique(int &np) {
25     sort(&P[0], &P[np], normal);
26     np = unique(&P[0], &P[np], issame) - P;
27 }
28 void convexHull(int &np, int &nc) {
29     int i, j, pos = 0;
30     for(i = 1; i < np; i++){
31         if(P[i].y<P[pos].y || (P[i].y==P[pos].y && P[i].x<P[pos].x))
32             pos = i;
33     }
34     swap(P[0], P[pos]); P0 = P[0]; sort(&P[1], &P[np], comp);
35     j = np - 1, i = np - 1;
36     while(j > 0 && triArea2(P0, P[j], P[j-1]) == 0)j--;
37     while(i > j){
38         swap(P[i], P[j]); i--,j++;
39     }
40     for(i = 0; i < 3; i++) C[i] = P[i];
41     for(i = j = 3; i < np; i++) {
42         while(triArea2(C[j-2], C[j-1], P[i]) < 0) j--;
43         C[j++] = P[i];
44     }
45     nc = j;
46 void compress(int &nc) {
47     int i, j, d;
48     C[nc] = C[0];
49     for(i=j=1; i < nc; i++) {
50         d = triArea2(C[j-1], C[i], C[i+1]);

```

```

51         if(d || (!d && issame(C[j-1], C[i+1]))) C[j++] = C[i];
52     }
53     nc = j;
54 }

```

Circle Intersection

```

1 dl commonArea(const Circle &a, const Circle &b) {
2     int dsq = sqDist(a, b); double d = sqrt((double)dsq);
3     if(sq(a.r + b.r) <= dsq) return 0;
4     if(a.r >= b.r && sq(a.r-b.r) >= dsq) return PI * b.r * b.r;
5     if(a.r <= b.r && sq(b.r-a.r) >= dsq) return PI * a.r * a.r;
6     double angleA = 2.0 * acos((a.r * a.r + dsq - b.r * b.r) / (2.0
7         * a.r * d));
8     double angleB = 2.0 * acos((b.r * b.r + dsq - a.r * a.r) / (2.0
9         * b.r * d));
10    return 0.5 * (a.r * a.r * (angleA - sin(angleA)) + b.r * b.r *
11        (angleB - sin(angleB)));
12 }

```

Point in Polygon

```

1 bool inConvexPoly(int nc, const pt &p) {
2     int st = 1, en = nc - 1, mid;
3     while(en - st > 1) {
4         mid = (st + en)>>1;
5         if(direction(C[0], C[mid], p) < 0) en = mid;
6         else st = mid;
7     }
8     if(direction(C[0], C[st], p) < 0) return false;
9     if(direction(C[st], C[en], p) < 0) return false;
10    if(direction(C[en], C[0], p) < 0) return false;
11    return true;
12 }//Ray casting algo to check point in convex -> start here
13 for(int i = 0; i < n; ++i){// Check if the pt is on any boundary
14     edge
15     pt a = polygon[i];
16     pt b = polygon[(i + 1) % n];
17     if(onSegment(a, b, p)) {return "BOUNDARY";}
18 }// Ray casting algorithm ;int count = 0;
19 for(int i = 0; i < n; ++i) {
20     pt a = polygon[i];pt b = polygon[(i + 1) % n];
21     if ((a.y > p.y) != (b.y > p.y)) {// Check if the pt is between
22         the y-coordinates of the edge
23         // Calculate the x-intersection of the edge with the horizontal
24         ray
25         double xIntersect = (double)(b.x - a.x) * (p.y - a.y) / (b.y -
26             a.y) + a.x;
27         if(p.x <= xIntersect) {count++;}
28     }
29 }if(count % 2 == 1){return "INSIDE";}else {return "OUTSIDE";}

```


Segment Intersection

```

1 struct pt { // Segment Intersection check
2     int x, y; pt() : x(0), y(0) {} pt(int a, int b) : x(a), y(b) {}
3 };
4 // Function to calculate the cross product (direction) of vectors (p1p2) and (p1p3)
5 int direction(const pt &p1, const pt &p2, const pt &p3) {
6     return (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
7 }
8 // Function to check if pt p3 lies on the segment p1p2
9 bool onsegment(const pt &p1, const pt &p2, const pt &p3) {
10     return min(p1.x, p2.x) <= p3.x && p3.x <= max(p1.x, p2.x) &&
        min(p1.y, p2.y) <= p3.y && p3.y <= max(p1.y, p2.y);
11 }
12 // Function to check if two segments (p1p2) and (p3p4) intersect
13 inline bool intersect(const pt &p1, const pt &p2, const pt &p3,
        const pt &p4) {
14     int d1, d2, d3, d4;
15     d1 = direction(p3, p4, p1); // Direction of (p3, p4, p1)
16     d2 = direction(p3, p4, p2); // Direction of (p3, p4, p2)
17     d3 = direction(p1, p2, p3); // Direction of (p1, p2, p3)
18     d4 = direction(p1, p2, p4); // Direction of (p1, p2, p4)
19     // General case: Segments (p1p2) and (p3p4) intersect if
        directions differ
20     if (((d1 < 0 && d2 > 0) || (d1 > 0 && d2 < 0)) && ((d3 < 0 &&
        d4 > 0) || (d3 > 0 && d4 < 0))) return true;
21     // Special cases: Check if the pts are collinear and lie on the
        segment
22     if (!d3 && onsegment(p1, p2, p3)) return true; // p3 is on
        segment (p1p2)
23     if (!d4 && onsegment(p1, p2, p4)) return true; // p4 is on
        segment (p1p2)
24     if (!d1 && onsegment(p3, p4, p1)) return true; // p1 is on
        segment (p3p4)
25     if (!d2 && onsegment(p3, p4, p2)) return true; // p2 is on
        segment (p3p4)
26     return false;
27 } // Segment intersection point start here
28 pair<bool, pt> doSegmentsIntersect(const pt &p1, const pt &p2, const
        pt &p3, const pt &p4) {
29     d1 d1 = direction(p3, p4, p1); d1 d2 = direction(p3, p4, p2);
30     d1 d3 = direction(p1, p2, p3); d1 d4 = direction(p1, p2, p4);
31     if (d1 * d2 < 0 && d3 * d4 < 0) { // Cramer's Rule.
32         d1 det = (p1.x - p2.x) * (p3.y - p4.y) - (p1.y - p2.y) * (
            p3.x - p4.x);
33         d1 ix = ((p1.x*p2.y - p1.y*p2.x)*(p3.x - p4.x) - (p1.x - p2
            .x)*(p3.x*p4.y - p3.y*p4.x)) / det;
34         d1 iy = ((p1.x*p2.y - p1.y*p2.x)*(p3.y - p4.y) - (p1.y - p2
            .y)*(p3.x*p4.y - p3.y*p4.x)) / det;
35         return {true, pt(ix, iy)};
36     } // Special cases - endpts on other segment

```

```

37     if (d1 == 0 && onSegment(p3, p4, p1)) return {true, p1};
38     if (d2 == 0 && onSegment(p3, p4, p2)) return {true, p2};
39     if (d3 == 0 && onSegment(p1, p2, p3)) return {true, p3};
40     if (d4 == 0 && onSegment(p1, p2, p4)) return {true, p4};
41     return {false, pt(0,0)};
42 }

```

Polygon Area

```

1 int area = 0, b = 0;
2 for(int i = 0; i < n; i++){
3     Point l = p[i], r = p[i+1]; area += (l.x * r.y - r.x * l.y); l.x
        -= r.x, l.y -= r.y;
4     int g = gcd(l.x, l.y); b += abs(g);}
5 peaks theorem a = abs(area) - b + 2; area = abs(area) / 2; // same
        same but
6 A=I+2 B 1 , A = area of polygone, I = inside point, B = Boundary
        point; p[n] = p[0]; int area = 0, b = 0;
7 for(int i = 0; i < n; i++){
8     Point l = p[i], r = p[i+1]; area += (l.x * r.y - r.x * l.y);
9     l.x -= r.x, l.y -= r.y; int g = gcd(l.x, l.y); b += abs(g);
10 } int a = abs(area) - b + 2;

```

Distance from Point to line Segment

```

1 // Function to calculate the distance from point C to line segment
        AB
2 double linePointDist(int A[], int B[], int C[], bool isSegment) {
3     int ABx = B[0] - A[0], ABx = B[1] - A[1]; int ACx = C[0] - A[0],
        ACy = C[1] - A[1];
4     int BCx = C[0] - B[0], BCy = C[1] - B[1]; double crossProd = ABx
        * ACy - ABx * ACx;
5     double ABLength = sqrt(ABx * ABx + ABx * ABx); double dist =
        fabs(crossProd) / ABLength;
6     if (isSegment) {
7         if ((ABx * BCx + ABx * BCy) > 0) return sqrt(BCx * BCx +
            BCy * BCy); // Distance to B
8         if ((-ABx * ACx - ABx * ACy) > 0) return sqrt(ACx * ACx +
            ACy * ACy); // Distance to A
9     } return abs(dist); }

```

Circle from 3 Points

```

1 // Function to find the center of the circle given 3 points
2 pair<d1, d1> findCircleCenter(d1 x1, d1 y1, d1 x2, d1 y2, d1 x3, d1
        y3) {
3     d1 mx1 = (x1 + x2) / 2, my1 = (y1 + y2) / 2, mx2 = (x2 + x3) /
        2, my2 = (y2 + y3) / 2;
4     d1 A1 = y2 - y1, B1 = x1 - x2, A2 = y3 - y2, B2 = x2 - x3; d1
        det = A1 * B2 - A2 * B1;
5     d1 cx = (B2 * (A1 * mx1 - A2 * mx2) - B1 * (A1 * mx2 - A2 * mx1
        )) / det;

```



```

6     dl cy = (A1 * (B2 * my2 - B1 * my1) - A2 * (B1 * my1 - B2 * my2
    )) / det;
7     return {cx, cy};
8 }

```

Advanced Algorithms

Heavy-Light Decomposition

```

1 const ll N = 1e5 + 123; const ll M = 4*N + 5; vi graph[N]; ll tree[M];
2 int Sz[N], heavy[N], Par[N], lev[N]; int Head[N], pos[N], idx = 1, ar[N],
    val[N];
3 void dfs(int node, int pa){
4     Sz[node] = 1; Par[node] = pa;
5     for(auto u : graph[node]){
6         if(u == pa) continue;
7         lev[u] = lev[node] + 1; dfs(u, node); Sz[node] += Sz[u];
8         if(Sz[u] > Sz[heavy[node]] || heavy[node] == 0){
9             heavy[node] = u;
10        }
11    }
12 }
13 void dfsHLD(int node, int chain){
14     Head[node] = chain; ar[idx] = val[node]; pos[node] = idx; idx++;
15     if(heavy[node] != 0){
16         dfsHLD(heavy[node], chain);
17     }
18     for(auto u : graph[node]){
19         if(u == Par[node]) continue;
20         if(heavy[node] != u){ dfsHLD(u, u); }
21     }
22 }
23 int Query(int a, int b){
24     int mx = 0;
25     while(Head[a] != Head[b]){
26         if(lev[Head[a]] < lev[Head[b]]){ swap(a, b); }
27         mx = max(mx, query(1, 1, idx, pos[Head[a]], pos[a]));
28         a = Par[Head[a]];
29     }
30     if(lev[a] < lev[b]) swap(a, b); int l = pos[b] + 1, r = pos[a];
31     if(l <= r) mx = max(mx, query(1, 1, idx, l, r)); // edge base b+1 to a,
        node base b to a
32     return mx;
33 } // main function start->
34 int n, q; cin >> n;
35 for(int i = 1; i <= n; i++){
36     graph[i].clear(); Sz[i] = 0; lev[i] = 0; heavy[i] = 0; pos[i] = 0,
        val[i] = 0; Head[i] = 0, Par[i] = 0;
37 }
38 map<int, pair<int, int>> bd; vector<tu> edge;
39 for(int i = 1; i < n; i++){
40     int u, v, w; cin >> u >> v >> w; graph[u].push_back(v); graph[v].

```

```

        push_back(u);
41     bd[i] = {u, v}; edge.push_back({u, v, w}); }
42     idx = 1; dfs(1, 0); val[1] = 0;
43     for(auto u : edge){ int a = u[0]; if(lev[u[0]] < lev[u[1]]) a = u[1];
        val[a] = u[2]; }
44     dfsHLD(1, 1); init(1, 1, idx);
45     while(1){ string s; cin >> s; if(s == "DONE") break;
46         if(s == "CHANGE"){
47             int i, v; cin >> i >> v; pair<int, int> x = bd[i]; int a = x.F;
48             if(lev[x.F] < lev[x.S]) a = x.S; int del = pos[a]; upgrade
                (1, 1, idx, del, v); }
49         else{ int a, b; cin >> a >> b; cout << Query(a, b) << endl; }
50     }

```

Mo's Algorithm

```

1 int ar[N], ans, n, answers[N], cnt[N]; int block_size;
2 void remove(int idx){ cnt[ar[idx]]--; if(cnt[ar[idx]] == 0) ans--; }
3 void add(int idx){ cnt[ar[idx]]++; if(cnt[ar[idx]] == 1) ans++; }
4 struct Query{
5     int l, r, idx;
6     bool operator < (const Query &x) const {
7         if(l / block_size == x.l / block_size) return r > x.r;
8         return l / block_size < x.l / block_size; }
9 }; // main function
10 void Goriber_solve(){ cin >> n; for(int i = 0; i < n; i++){ cin >> ar
    [i]; } int q; cin >> q;
11     block_size = (int) sqrt(n + .0) + 1; vector<Query> queries;
12     for(int i = 0; i < q; i++){ int l, r; cin >> l >> r; queries.pb({l,
        r, i}); }
13     sort(all(queries)); int cur_l = 0; int cur_r = -1;
14     for (Query &q : queries) { q.l--; q.r--;
15         while (cur_l > q.l) { cur_l--; add(cur_l); }
16         while (cur_r < q.r) { cur_r++; add(cur_r); }
17         while (cur_l < q.l) { remove(cur_l); cur_l++; }
18         while (cur_r > q.r) { remove(cur_r); cur_r--; }
19         answers[q.idx] = ans; }
20     for(int i = 0; i < q; i++){
21         cout << answers[i] << endl; } }

```

Square Root Decomposition

```

1 int n; vector<int> a(n); int len = (int) sqrt(n + .0) + 1; vector<
    int> b(len);
2 for(int i = 0; i < n; ++i) b[i / len] += a[i];
3 while(q--){
4     int l, r; int sum = 0;
5     for(int i = l; i <= r; ){
6         if(i % len == 0 && i + len - 1 <= r) {
7             sum += b[i / len]; i += len; }
8         else { sum += a[i]; ++i; }
9     }
10 }

```

Euler Tour

```
1 vector<int> graph[N];vector<int> flat_tree;map<int,pair<int,int>>
  mp;int cnt = 0;
2 void dfs(int node,int par) {
3   flat_tree.push_back(node);mp[node].first = cnt;cnt++;
4   for(auto child : graph[node]) {
5     if(child == par)continue;dfs(child,node);
6   }
7   mp[node].second = cnt;flat_tree.push_back(node);cnt++;}
```

Tree Diameter

```
1 vi graph[N];
2 void dfs(int node,int par,int lev,vi &d){ d[node] = lev;
3   for(auto u : graph[node]){
4     if(u != par){ dfs(u,node,lev+1,d);}
5   }
6 }
7 void Goriber_solve(){
8   int n;cin >> n;for(int i = 0; i <= n; i++){graph[i].clear();}
9   for(int i = 1; i < n; i++){
10     int u,v;cin >> u >> v;u--,v--;graph[u].pb(v);graph[v].pb(u);
11   }
12   vi d1(n,0),d2(n,0);dfs(0,-1,0,d1);int a = max_element(all(d1))
    - d1.begin();dfs(a,-1,0,d1);
13   int b = max_element(all(d1)) - d1.begin();dfs(b,-1,0,d2);}
```

2D Algorithms

2D Pre Sum + mat Rotation + grid 8 direction move

```
1 pre[i][j] = ar[i][j] + pre[i-1][j] + pre[i][j-1] - pre[i-1][j-1];
2 cout<< pre[c][d] - pre[a-1][d] - pre[c][b-1] + pre[a-1][b-1] ;
3 for(int i = 0; i < n; i++) {
4   for(int j = 0; j < n; j++) {res[j][n - i - 1] = mat[i][j];}
5 }
6 int fx[] = {+0,+0,+1,-1,-1,+1,-1,+1};int fy[] =
    {-1,+1,+0,+0,+1,+1,-1,-1};
```

Spiral Matrix

```
1 int x = (n + 1) / 2, y = (n + 1) / 2;int num = 0;ans[x][y] = 0;// 1
    based matrix
2 for (int i = 2; i <= n; ++i) {
3   if (i % 2 == 0) { ans[x][++y] = ++num;
4     for (int j = 1; j < i; ++j)ans[++x][y] = ++num;
5     for (int j = 1; j < i; ++j)ans[x][--y] = ++num;
6   }
7   else {ans[x][--y] = ++num;
8     for (int j = 1; j < i; ++j)ans[--x][y] = ++num;
```

```
9   for (int j = 1; j < i; ++j)ans[x][++y] = ++num;
10 }
```

Matrix Exponentiation

```
1 vector<vi>matMul(vector<vi>&mat1,vector<vi>&mat2,int n){ vector<vi>
  res(n, vi(n,0));
2   for(int i = 0; i < n; i++){
3     for(int j = 0; j < n; j++){
4       for(int k = 0; k < n; k++){res[i][j]=(res[i][j]+((ll)mat1[i]
        ][k]*mat2[k][j])%mod)%mod;}
5     }
6   } return res;
7 }
8 vector<vi>matExpo(vector<vi>& mat,int k,int n){
9   if(k == 1)return mat; vector<vi>res=matExpo(mat,k/2,n); res =
    matMul(res,res,n);
10  if(k & 1)res=matMul(res,mat,n);
11  return res;
12 }
13 F(k+1) =|mat^k|F(k) |
14 F(k) = | |F( k 1 )|
15 S(k) = | |S( k 1 )|
16 S( k 1 ) =| |S( k 2 )|
17 vi mulBase(vector<vi>&mat,vi &base,int n){ vi res(n,0);
18   for(int i = 0; i < n; i++){
19     for(int j = 0; j < n; j++){
20       res[i] += (ll)mat[i][j] * base[j] % mod;res[i] %= mod;
21     }
22   } return res;
23 }
```

Utilities

Path Compression

```
1 vi a(n), b(n);sort(all(b));b.erase(unique(all(b)), b.end());
2 for(int i = 0; i < n; i++){a[i] = lower_bound(all(b), a[i]) - b.
    begin() + 1;}
```

Histogram

```
1 int getMaxArea(vector<int>&arr, int n) {
2   stack<int> s;s.push(-1);int area = arr[0];int i = 0;vector<int>
    left(n, -1), right(n, n);
3   while(i < n) {
4     while(!s.empty() && s.top() != -1 && arr[s.top()] > arr[i]) {
5       right[s.top()] = i;s.pop();
6     }
7     if(i > 0 && arr[i] == arr[i - 1]) {
8       left[i] = left[i - 1];
9     }else { leftr[i] = s.top();}
```

```

10     s.push(i);i++;
11 }
12 for (int j = 0; j < n; j++) {
13     area = max(area, arr[j] * (right[j] - left[j] - 1));
14 } return area;
15 }

```

Is Intersect All Points

```

1 auto f = [&] (dl m) -> bool {vector<pair<dl,dl>> p;
2     for(int i = 0; i < n; i++) {
3         dl r = x[i] + m*v[i];dl l = x[i] - m*v[i];p.push_back({l,r});
4     }
5     dl l = -INF,r = INF;
6     for(int i = 0; i < n; i++) {
7         dl left = p[i].F;dl right = p[i].S;
8         if(l > right || r < left) {return 0;} l = max(l,left); r = min(
9             r,right);
10    } return 1;
11 };

```

Unique Line Equations

```

1 tuple<int,int,int> norm(int a, int b, int c) {
2     int d = (__gcd(a,__gcd(b,c))); a/=d; b/=d; c/=d;int s = a?a:b?b:c
3     ;
4     if(s < 0) a=-a, b=-b, c=-c;
5     return {a,b,c};
6 }
7 int countLines(vector<pair<int,int>>& p) {
8     set<tuple<int,int,int>> s;
9     for(int i=0; i<p.size(); i++) {
10        for(int j=i+1; j<p.size(); j++) {
11            auto [x1,y1]=p[i], [x2,y2]=p[j]; int a=y2-y1, b=x1-x2, c=x2*
12                y1-x1*y2; s.insert(norm(a,b,c));
13        }
14    }return s.size();
15 }

```

Special Problems

Kth Number L to R Using Persistent Seg Tree

```

1 //number of elements greater than k in the subsequence ai, ai+1 ...aj
2 struct node{
3     int val;
4     node* lc;
5     node* rc;
6 };
7 int ar[N];
8 node* versions[N+5];

```

```

9 vector<node*> lagbe;
10
11 node* build(int l,int r){
12     if(l == r){
13         node* me = new node();
14         lagbe.pb(me);
15         me->val = 0;
16         me->lc = NULL;
17         me->rc = NULL;
18         return me;
19     }
20     node* my = new node();
21     lagbe.pb(my);
22     int mid = (l + r)/2;
23     my->lc = build(l,mid);
24     my->rc = build(mid+1,r);
25     my->val = (my->lc->val + my->rc->val);
26     return my;
27 }
28 int query(node* n1,node* n2,int l,int r,int L,int R){
29     if(n1 == NULL || n2 == NULL)return 0;
30     if(R < l || r < L)return 0;
31     if(L <= l && r <= R){
32         return n1->val - n2->val;
33     }
34     int mid = (l + r)/2;
35     int p = query(n1->lc,n2->lc,l,mid,L,R);
36     int q = query(n1->rc,n2->rc,mid+1,r,L,R);
37     return p + q;
38 }
39 node* update(node* my,int l,int r,int ind,int val){
40     if(l == r){
41         node* new_my = new node();
42         lagbe.pb(new_my);
43         new_my->lc = NULL;
44         new_my->rc = NULL;
45         new_my->val = my -> val + val;
46         return new_my;
47     }
48     node* new_me = new node();
49     lagbe.pb(new_me);
50     int mid = (l + r)/2;
51     if(ind <= mid){
52         node* x = update(my->lc,l,mid,ind,val);
53         new_me->lc = x;
54         new_me->rc = my->rc;
55     }
56     else{
57         node* y = update(my->rc,mid+1,r,ind,val);
58         new_me->lc = my->lc;
59         new_me->rc = y;
60     }
61     new_me->val = new_me->lc->val + new_me->rc->val;

```

```

62     return new_me;
63 }
64 vi com;
65 void Goriber_solve()
66 {
67     int n,q;
68     cin >> n;
69     for(int i = 1; i <= n; i++){
70         cin >> ar[i];
71         com.pb(ar[i]);
72     }
73     sort(all(com));
74     com.resize(unique(all(com))-com.begin());
75     map<int,int> pos;
76     int x = 0;
77     for(auto u : com)pos[u] = x++;
78     int si = sz(com)+1;
79     versions[0] = build(0,si);
80     for(int i = 1; i <= n; i++){
81         versions[i] = update(versions[i-1],0,si,pos[ar[i]],1);
82     }
83     int ans = 0;
84     cin >> q;
85     while(q--){
86         int a,b,c;
87         cin >> a >> b >> c;
88         a = (a ^ ans);
89         b = (b ^ ans);
90         c = (c ^ ans);
91         int val = upper_bound(all(com),c)-com.begin();
92         if(a <= 0)a = 1;
93         if(b > n)b = n;
94         if(a > b){
95             cout << 0 << endl;
96             ans = 0;
97             continue;
98         }
99         ans = query(versions[b],versions[a-1],0,si,val,si);
100         cout << ans << endl;
101     }
102     for(auto u : lagbe){
103         delete u;
104     }
105 }

```

How Many Subarrays XOR < k

```

1 class Trie{
2     int cnt(int num, int k){
3         Node* cur = root;int ret = 0;
4         for(int i = 31; i >= 0; i--){
5             int num1 = (num >> i) & 1;int num2 = (k >> i) & 1;
6             if(num2){

```

```

7                 if(cur -> contains(num1)){ ret += cur -> get(num1)
8                     -> getPre();}
9                 if(cur -> contains(num1 ^ 1))cur = cur -> get(num1
10                     ^ 1);
11                 else return ret;
12             }
13             else{
14                 if(cur -> contains(num1))cur = cur -> get(num1);
15                 else return ret;
16             }
17         }
18     }return ret;
19 };// main function trie.insert(0); xr ^= ar[i]; ans += trie.cnt(xr,
20     k);trie.insert(xr);

```

Min Length with Max XOR $\geq k$

```

1 class Trie {
2     int getMax(int num,int lo) {
3         Node* cur_node = root;int mx = 0;
4         for(int i = 31; i >= 0; i--) {
5             int bit = (num >> i) & 1;int opo = bit ^ 1;
6             if(cur_node -> contains(opo) && cur_node -> get(opo) ->
7                 mx_index >= lo) {
8                 mx = ((1LL << i) | mx);cur_node = cur_node -> get(opo);
9             }
10            else {cur_node = cur_node -> get(bit);}
11        }
12        return mx;
13    };// main function -> int n,k; cin >> n >> k; int ans = n + 1; Trie
14    trie;
15    for(int i = 1; i <= n; i++) { int x; cin >> x; trie.insert(x,i);
16        int lo = 1, hi = i,cnt = -1;
17        while(lo <= hi) { int mid = (lo + hi) / 2;
18            if((trie.getMax(x,mid)) >= k) { cnt = mid; lo = mid + 1;}
19            else hi = mid - 1;
20        } if(cnt != -1)ans = min(ans,i + 1 - cnt);
21    } if(ans == n + 1)cout << -1 << endl; else cout << ans << endl;

```

$$I < J < K \text{ and } a_i > a_j > a_k$$

```

1 vi a(n), b(n); for(int i = 0; i < n; i++) { cin >> a[i]; b[i] = a[i]
2     ]; }
3 sort(all(b)); b.erase(unique(all(b)), b.end());
4 for(int i = 0; i < n; i++) { a[i] = lower_bound(all(b), a[i]) - b.
5     begin() + 1; } vi L(n), R(n);
6 for(int i = 0; i < n; i++) {
7     L[i] = sum(a[i]+1,n+1); update(a[i],1); mem(tree,0);// use BIT}
8     for(int i = n-1; i >= 0; i--) {
9         R[i] = sum(1,a[i]-1); update(a[i],1);
10    } int ans = 0;

```

```

9   for(int i = 0; i < n; i++) {
10       ans += L[i] * R[i];
11   } cout << ans << endl;}

```

Tetrahedron Formulas

```

1  glass area = ((pi*h)/3) *((r1*r1)+(r2*r2)+(r1*r2)) //Some
   tetrahedron formulas
2  inline double volume(double u, double v, double w, double U, double
   V, double W) {
3      double u1,v1,w1; u1 = v * v + w * w - U * U;
4      v1 = w * w + u * u - V * V; w1 = u * u + v * v - W * W;
5      return sqrt(4.0*u*u*v*v*w*w - u*u*u1*u1 - v*v*v1*v1 - w*w*w1*w1
   + u1*v1*w1) / 12.0; }
6  double surface(double a, double b, double c) {
7      return sqrt((a + b + c) * (-a + b + c) * (a - b + c) * (a + b -
   c)) / 4.0; }
8  double insphere(double WX, double WY, double WZ, double XY, double
   XZ, double YZ) {
9      double sur, rad;
10     sur = surface(WX, WY, XY) + surface(WX, XZ, WZ) + surface(WY,
   YZ, WZ) + surface(XY, XZ, YZ);
11     rad = volume(WX, WY, WZ, YZ, XZ, XY) * 3.0 / sur; return rad
   ;}

```

Templates

Template with PBDS

```

1  #include <ext/pb_ds/assoc_container.hpp>
2  #include <ext/pb_ds/tree_policy.hpp>
3  using namespace __gnu_pbds;template <class T>
4  using ordered_set = tree<T,null_type,less<T>, rb_tree_tag,
   tree_order_statistics_node_update>;
5  ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
6  #define endl '\n' #define F first #define S second
7  #define all(s) (s).begin(), (s).end()
8  #define rall(s) s.rbegin(), s.rend()
9  #define sz(x) (int)x.size()
10 #define mem(a, b) memset(a, b, sizeof(a))
11 #define cnt_bit(x) __builtin_popcountll(x)
12 #define getbit(n, i) (((n) & (1LL << (i))) != 0)
13 #define msb(x) 31 - __builtin_clz(x)
14 typedef array<int,3> tu;
15 mt19937_64 gen(random_device{}());int r_num =
   uniform_int_distribution<long long>{0, 1e18}(gen);
16 view->layout 3 column then group max 2 column
17 Sublime text setup ->tool->build system->newbuild
18 {
19     "shell_cmd": "g++ \"${file}\" -o a.exe && a.exe < input.in >
   output.in"
20 }

```

```

21 {
22     "cmd": [
23         "g++ -std=c++20 \"${file_name}\" -o \"${file_base_name}\" &&
   timeout 4s ./\"${file_base_name}\" < input.txt > output.
   txt"
24     ],
25     "selector": "source.c++",
26     "shell": true,
27     "working_dir": "${file_path}"
28 }

```

GP Hash Table

```

1  struct custom_hash {
2      static uint64_t splitmix64(uint64_t x) {
3          x += 0x9e3779b97f4a7c15;
4          x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
5          x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
6          return x ^ (x >> 31);
7      }
8      size_t operator()(uint64_t x) const {
9          static const uint64_t FIXED_RANDOM = chrono::steady_clock::now
   ().time_since_epoch().count();
10         return splitmix64(x + FIXED_RANDOM);
11     }
12     size_t operator()(const pair<int, int>&x) const {
13         return ((long long)x.first) ^ (((long long)x.second) << 32);
14     }
15 }; unordered_map<pair<ll, ll>, vector<ll>, custom_hash> bd;
16 unordered_map<pair<ll, ll>, ll, custom_hash> dp;

```

Saad Segment Tree

```

1  struct ST {
2      struct Node { // change here
3          long long sum = 0; Node () {}
4      };
5      vector<long long> a;
6      vector<Node> t;
7      Node op (Node x, Node y) { // change here
8          x.sum += y.sum;
9          return x;
10     }
11     ST (vector<long long> &A) { // change here
12         a = A;
13         t.resize (4 * a.size());
14         bld (1, 0, a.size() - 1);
15     }
16     void bld (int i, int l, int r) {
17         if (l == r) {
18             t[i].sum = a[l];
19             return;
20         }

```

```

21     int m = (l + r) / 2;
22     bld (i * 2, l, m);
23     bld (i * 2 + 1, m + 1, r);
24     t[i] = op (t[i * 2], t[i * 2 + 1]);
25 }
26 void upd (int i, int l, int r, int p, int v) {
27     if (l == r) {
28         a[p] += v;                // change here
29         t[i].sum = a[p];
30         return;
31     }
32     int m = (l + r) / 2;
33     if (p <= m) upd (i * 2, l, m, p, v);
34     else upd (i * 2 + 1, m + 1, r, p, v);
35     t[i] = op (t[i * 2], t[i * 2 + 1]);
36 }
37 Node qry (int i, int l, int r, int L, int R) {
38     if (R < l || r < L) return Node();
39     if (L <= l && r <= R) return t[i];
40     int m = (l + r) / 2;
41     return op (qry (i * 2, l, m, L, R),
42               qry (i * 2 + 1, m + 1, r, L, R));
43 }
44 };
45 struct ST {
46     struct N {
47         long long s = 0, m = 1e18;
48     };
49     vector<N> t;
50     vector<long long> a, lz;
51     int n;
52     N op (N x, N y) {
53         x.s += y.s;
54         x.m = min (x.m, y.m);
55         return x;
56     }
57     void push (int i, int l, int r, long long v) {
58         t[i].s = v * (r - l + 1);
59         t[i].m = v;
60         if (l != r) lz[i * 2] = lz[i * 2 + 1] = v;
61         lz[i] = 0;
62     }
63     void build (int i, int l, int r) {
64         if (l == r) {
65             t[i].s = a[l];
66             t[i].m = a[l];
67             return;
68         }
69         int m = (l + r) / 2;
70         build (i * 2, l, m);
71         build (i * 2 + 1, m + 1, r);
72         t[i] = op (t[i * 2], t[i * 2 + 1]);
73     }

```

```

74 void upd (int i, int l, int r, int L, int R, long long v) {
75     if (lz[i]) push (i, l, r, lz[i]);
76     if (r < L || R < l) return;
77
78     if (L <= l && r <= R) {
79         push (i, l, r, v);
80         return;
81     }
82
83     int m = (l + r) / 2;
84     upd (i * 2, l, m, L, R, v);
85     upd (i * 2 + 1, m + 1, r, L, R, v);
86     t[i] = op (t[i * 2], t[i * 2 + 1]);
87 }
88 N qry (int i, int l, int r, int L, int R) {
89     if (r < L || R < l) return N();
90     if (lz[i]) push (i, l, r, lz[i]);
91     if (L <= l && r <= R) return t[i];
92
93     int m = (l + r) / 2;
94     return op (qry (i * 2, l, m, L, R), qry (i * 2 + 1, m + 1,
95                                               r, L, R));
96 }
97 ST (vector<long long> &A) {
98     a = A;
99     n = A.size();
100    t.resize (4 * n);
101    lz.assign (4 * n, 0);
102    build (1, 0, n - 1);
103 }
104 void InUpdate (int i, long long v) {
105     upd (1, 0, n - 1, i, i, v);
106 }
107 void R_Update (int L, int R, long long v) {
108     upd (1, 0, n - 1, L, R, v);
109 }
110
111 N query (int L, int R) {
112     return qry (1, 0, n - 1, L, R);
113 }
114 };

```

Kaium Segment Tree

```

1 struct Node { // Maximum Subarray Sum
2     int sum;
3     int sub; // Value of interest
4     int pre;
5     int suf;
6
7     Node(): sum(0), sub(-1e18), pre(sub), suf(sub) {}
8     Node(int x): sum(x), sub(x), pre(x), suf(x) {}

```

```

9   operator int() { return sub; }
10
11   Node operator+(const Node& b) {
12       Node c;
13       c.sum = sum + b.sum;
14       c.sub = max({ sub, b.sub, suf+b.pre });
15       c.pre = max(pre, sum+b.pre);
16       c.suf = max(b.suf, suf+b.sum);
17       return c;
18   }
19 };
20
21 struct SegTree {
22     int n;
23     vector<Node> Tree;
24     SegTree(int size) {
25         // Round up to closest power of 2
26         n = 1 << (32 - __builtin_clz(--size));
27         Tree.resize(2*n);
28     }
29
30     // Constructor to initialize by array
31     SegTree(const VI& A): SegTree(A.size()) {
32         for (int i = 0; i < A.size(); i++)
33             Tree[n + i] = Node(A[i]);
34         // Build the segment tree
35         ROF (i, 1, n-1)
36             Tree[i] = Tree[2*i] + Tree[2*i+1];
37     }
38     // Update value at position p
39     void update(int p, int value) {
40         p += n;
41         Tree[p] = Node(value);
42         for (p /= 2; p > 0; p /= 2)
43             Tree[p] = Tree[2*p] + Tree[2*p+1];
44     }
45     // Query on interval [l, r)
46     Node query(int l, int r) {
47         Node resL, resR;
48         for (l += n, r += n; l < r; l /= 2, r /= 2) {
49             if (l % 2) resL = resL + Tree[l++];
50             if (r % 2) resR = Tree[--r] + resR;
51         } // Non-commutative merge
52         return resL + resR;
53     }
54 };

```

Kaium I/O

```

1 #define DEBUG
2 #define LOCAL
3 // #define TCASE
4 // #define PB_DS

```

```

5
6 #include <bits/stdc++.h>
7 using namespace std;
8
9 #ifdef LOCAL
10 #define _LB_ cerr << "{ "
11 #define _RB_ cerr << " }"
12 #else
13 #define endl '\n'
14 #define _LB_
15 #define _RB_
16 #endif
17
18 #define _SP_ << &" "[i++] <<
19 #define _FN_OUT_ { _LB_; int i = 0; for (auto& a : A) out _SP_ a;
20 _RB_; return out; }
21 #define _FN_IN_ { for (auto& a : A) in >> a; return in; }
22 #define _OP_OUT_ ostream& operator<<(ostream& out, const
23 #define _OP_IN_ istream& operator>>(istream& in,
24
25 #define temT template <class T>
26 #define temTs template <class... Ts>
27 #define temTN template <class T, size_t N>
28
29 temTs _OP_OUT_ pair<Ts...>& p) { _LB_; out << p.first << " " << p.
30 second; _RB_; return out; }
31 temTs _OP_IN_ pair<Ts...>& p) { return in >> p.first >> p.second; }
32 temTN _OP_OUT_ array<T, N>& A) _FN_OUT_
33 temTN _OP_IN_ array<T, N>& A) _FN_IN_
34 #define def_out(cont) temT _OP_OUT_ cont<T>& A) _FN_OUT_
35 #define def_in(cont) temT _OP_IN_ cont<T>& A) _FN_IN_
36 def_in(vector) def_in(deque) def_out(vector) def_out(deque) def_out
37 (set) def_out(multiset)
38 temTs _OP_OUT_ map<Ts...>& A) _FN_OUT_
39
40 #ifdef PB_DS
41 #include <ext/pb_ds/assoc_container.hpp>
42 #include <ext/pb_ds/tree_policy.hpp>
43 using namespace __gnu_pbds;
44 #define _RBT_ rb_tree_tag, tree_order_statistics_node_update
45 temT using ordered_set = tree<T, null_type, less<T>, _RBT_>;
46 temT using ordered_multiset = tree<T, null_type, less_equal<T>,
47 _RBT_>;
48 def_out(ordered_set) def_out(ordered_multiset)
49 #endif
50
51 temTs istream& c_in(Ts&... args) { return ((cin >> args), ...); }
52 temTs ostream& c_out(const Ts&... args) { int i = 0; return ((cout
53 _SP_ args), ...) << endl; }
54 ostream& c_out(bool b) { return c_out(b ? "YES" : "NO"); }
55
56 #ifdef LOCAL

```

```

53 temTs ostream& c_err(const Ts&... args) { int i = 0; return ((cerr
    _SP_ args), ...) << endl; }
54 #else
55 #define c_err(...)
56 #endif
57 #define d_bug(args...) c_err(#args, '=', args)
58
59
60 #define all(A) A.begin(), A.end()
61 #define rall(A) A.rbegin(), A.rend()
62 #define sum_of(A) accumulate(all(A), 0ll)
63 #define min_of(A) *min_element(all(A))
64 #define max_of(A) *max_element(all(A))
65
66 #define int long long
67 #define uint unsigned int
68 using VI = vector<int>;
69 using VC = vector<char>;
70 using DI = deque<int>;
71 using QI = queue<int>;
72 using SI = set<int>;
73 using MII = map<int, int>;
74 using VVI = vector<VI>;
75 template<size_t N> using AI = array<int, N>;
76 using II = AI<2>; // instead of pair<int, int>
77 using III = AI<3>; // ... tuple<int, int, int>
78 using VII = vector<II>;
79 using VIII = vector<III>;
80 using VVII = vector<VII>;
81 template<class T> using PQ = priority_queue<T, vector<T>, greater<T>
    >>;
82 #define VMAT(n, m) VVI(n, VI(m))
83 #define VMATa(n, m, a) VVI(n, VI(m, a))
84 #define AMAT(n, m) vector<AI<m>>>(n)
85 #define FOR(i, l, r) for (int i = l; i <= r; i++)
86 #define ROF(i, l, r) for (int i = r; i >= l; i--)

```

Kaium Hashing

```

1 II operator+(II a, II b) { return { a[0] + b[0], a[1] + b[1] }; }
2 II operator-(II a, II b) { return { a[0] - b[0], a[1] - b[1] }; }
3 II operator*(II a, II b) { return { a[0] * b[0], a[1] * b[1] }; }
4 II operator%(II a, II b) { return { a[0] % b[0], a[1] % b[1] }; }
5
6 struct StrHash {
7     int n;
8     II base, mod;
9     vector<II> pref, inv;
10
11     StrHash(string s, II b = { 31, 37 }, \
12     II m = { 1000000007, 1000000009 }): n(s.size()), \
13     base(b), mod(m), pref(n+1), inv(n+1) {
14         II sum = { 0, 0 };

```

```

15     II pow = { 1, 1 };
16
17     for (int i = 0; i < n; i++) {
18         int c = s[i] - 'a' + 1;
19         sum = (sum + II{ c, c } * pow % mod) % mod;
20         pref[i+1] = sum;
21
22         inv[i][0] = bin_exp(pow[0], mod[0]-2, mod[0]);
23         inv[i][1] = bin_exp(pow[1], mod[1]-2, mod[1]);
24
25         pow = pow * base % m;
26     }
27 }
28
29 II substr_hash(int i, int j) { // [i, j)
30     i = max(0ll, min(n, i));
31     j = max(0ll, min(n, j));
32     II hash = mod + pref[j] - pref[i];
33     hash = hash % mod * inv[i] % mod;
34     return hash;
35 }
36 };

```

Min Cut

```

1 int n, m; c_in(n, m);
2 VII E(m); c_in(E);
3
4 VVI G(n+1);
5 auto M = VMAT(n+1, n+1);
6
7 for (auto [u, v]: E) {
8     G[u].push_back(v);
9     G[v].push_back(u);
10    M[u][v] = 1;
11    M[v][u] = 1;
12 }
13
14 auto max_flow = [&](int s, int t) {
15     VC Vis(n+1);
16     int delta = 1ll << 30;
17
18     auto dfs = [&](int u, int fu, auto& self) -> int {
19         if (u == t) return fu;
20         Vis[u] = true;
21
22         for (auto v : G[u]) if (!Vis[v] && M[u][v] >= delta) {
23             int pushed = self(v, min(fu, M[u][v]), self);
24             if (pushed) {
25                 M[u][v] -= pushed;
26                 M[v][u] += pushed;
27                 return pushed;
28             }

```



```

29     }
30     return 0;
31 };
32
33 int flow = 0;
34
35 while (delta) {
36     int new_flow;
37     do {
38         fill(all(Vis), false);
39         new_flow = dfs(s, INF, dfs);
40         flow += new_flow;
41     } while (new_flow);
42
43     delta >>= 1;
44 }
45 return flow;
46 };
47 c_out(max_flow(1, n));
48 VII Cuts;
49 VC Vis(n+1);
50
51 auto dfs_cuts = [&](int u, auto& self) -> void {
52     Vis[u] = true;
53     for (auto v : G[u])
54         if (!Vis[v] && M[u][v])
55             self(v, self);
56 };
57 dfs_cuts(1, dfs_cuts);

```

Convex Hull (kaium)

```

1 II operator-(II p1, II p2) \
2 { return { p1[0] - p2[0], p1[1] - p2[1] }; }
3 int orientation(II p1, II p2, II p) {
4     II v1 = p2 - p1, v2 = p - p1;
5     int cross = v1[0] * v2[1] - v1[1] * v2[0];
6     return (cross > 0) - (cross < 0);
7 }
8 deque<II> convex_hull(vector<II>& V, \
9 bool collinear = false) {
10     II lp = V[0]; // find the lowest point
11     for (auto& p : V) if (p[1] < lp[1] || \
12         (p[1] == lp[1] && p[0] < lp[0])) lp = p;
13
14     deque<array<double, 3>> D;
15     int n = V.size();
16     // generate angle, distance, index for sorting
17     FOR (i, 0, n-1) {
18         II v = V[i] - lp;
19         double angle = atan2(v[1], v[0]);
20         double dist2 = v[0] * v[0] + v[1] * v[1];
21         D.push_back({ angle, dist2, (double)i });

```

```

22     }
23     sort(all(D));
24     // reverse collinear points at the back
25     if (collinear) {
26         double angle = D.back()[0];
27         auto it = D.rbegin();
28         while (it < D.rend() && (*it)[0] == angle) it++;
29         reverse(D.rbegin(), it);
30     }
31     deque<II> H;
32     int i; // add first two points
33     i = D[0][2]; H.push_front(V[i]); D.pop_front();
34     i = D[0][2]; H.push_front(V[i]); D.pop_front();
35     for (auto& d : D) {
36         i = d[2];
37         II p = V[i];
38         // < for keeping collinear points, <= otherwise
39         while (orientation(H[1], H[0], p) < 0)
40             H.pop_front();
41         H.push_front(p);
42     }
43     reverse(all(H));
44     return H;
45 }

```